

# Algorithmische Bioinformatik 1

Dr. Hanjo Täubig

Lehrstuhl für Effiziente Algorithmen  
(Prof. Dr. Ernst W. Mayr)  
Institut für Informatik  
Technische Universität München

Sommersemester 2008



# Übersicht

- 1 Algorithmen zur Textsuche
  - Rückwärtssuche
  - Boyer-Moore-Algorithmus

## Zweiter naiver Ansatz

- weiterer Ansatzpunkt zum Suchen in Texten:  
das gesuchte Wort mit einem Textstück nicht mehr von links nach rechts, sondern **von rechts nach links** vergleichen
- Vorteil:  
In Alphabeten mit vielen verschiedenen Symbolen kann man bei einem Mismatch an der Position  $i + j$  in  $t$ ,  $i$  auf  $i + j + 1$  setzen, falls das im Text  $t$  vorgefundene Zeichen  $t_{i+j}$  gar nicht im gesuchten Wort  $s$  vorkommt.
- D.h. also, in so einem Fall kann man das Suchwort gleich um  $j + 1$  Positionen verschieben (im günstigsten Fall  $m$  Stellen).

## Zweiter naiver Ansatz

---

**Algorithmus 6** : `bool Naiv2(char t[], int n, char s[], int m)`

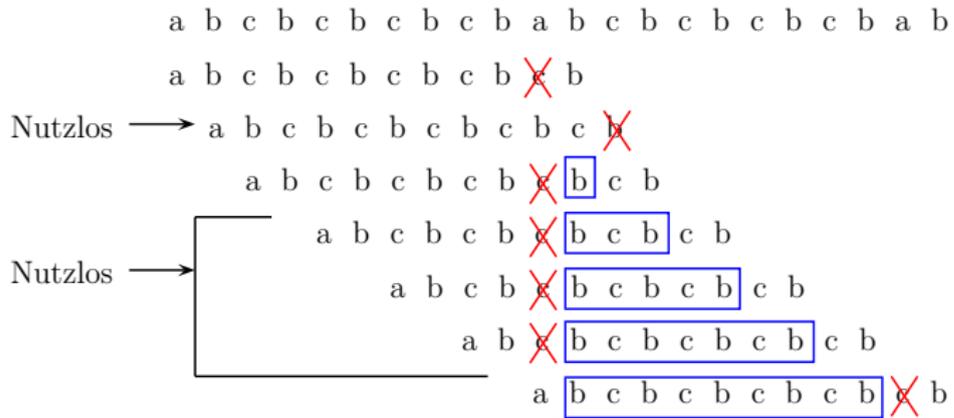
---

```
int i := 0;
int j := m - 1;
while i ≤ n - m do
  while t[i + j] = s[j] do
    if j = 0 then return TRUE;
    j--;
  i++;
  j := m - 1;
return FALSE;
```

---

Naive Methode mit rechts-nach-links Vergleichen  
(ohne größere Shifts)

## Nutzlose Verschiebungen

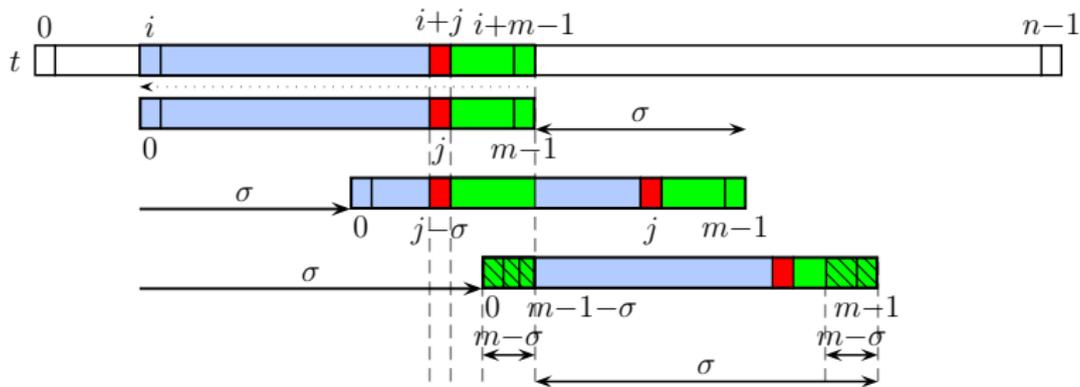


Übereinstimmung aufgrund eines zulässigen Shifts

# Nutzlose Verschiebungen

- Wie beim KMP-Algorithmus ist es nutzlos (siehe zweiter Versuch von oben), wenn man die Zeichenreihe so verschiebt, dass im Bereich erfolgreicher Vergleiche nach einem Shift keine Übereinstimmung mehr herrscht.
- Es macht ebenfalls keinen Sinn, das Muster so zu verschieben, dass an der Position des Mismatches in  $t$  im Muster  $s$  wiederum dasselbe Zeichen zum Liegen kommt, das schon vorher den Mismatch ausgelöst hat.  
(vierter bis sechster Versuch)

## Boyer-Moore-Algorithmus



Skizze: Zulässige Shifts bei Boyer-Moore (Strong-Good-Suffix-Rule)

# Boyer-Moore-Algorithmus: Shifts

- Zwei mögliche Arten eines „vernünftigen“ Shifts bei der Variante von Boyer-Moore:
  - Im oberen Teil ist ein „kurzer“ Shift angegeben, bei dem im grünen Bereich die Zeichen nach dem Shift weiterhin übereinstimmen.  
Das rote Zeichen in  $t$  (das den Mismatch ausgelöst hat) soll nach dem Shift auf ein anderes Zeichen in  $s$  treffen, damit überhaupt die Chance auf Übereinstimmung besteht.
  - Im unteren Teil ist ein „langer“ Shift angegeben, bei dem die Zeichenreihe  $s$  soweit verschoben wird, dass an der Position des Mismatches in  $t$  gar kein weiterer Vergleich mehr entsteht. Aber auch hier: wieder Übereinstimmung im schraffierten grünen Bereich mit den bereits verglichenen Zeichen aus  $t$

# Boyer-Moore-Algorithmus: Good-Suffix-Rule

- Kombination der beiden Regeln: **Good-Suffix-Rule**  
(da man darauf achtet, die Zeichenreihen so zu verschieben, dass im letzten übereinstimmenden Bereich nach dem Shift wieder Übereinstimmung herrscht)
- Achtet man noch speziell darauf, dass an der Position, in der es zum Mismatch gekommen ist, jetzt in  $s$  ein anderes Zeichen liegt als das, welches den Mismatch ausgelöst hat, so spricht man von der **Strong-Good-Suffix-Rule** (andernfalls Weak-Good-Suffix-Rule).
- Wir betrachten nur die Strong-Good-Suffix-Rule, da ansonsten die worst-case Laufzeit wieder quadratisch werden kann.

# Boyer-Moore-Algorithmus

---

**Algorithmus 7** : bool Boyer-Moore(char t[], int n, char s[], int m)

---

int S[m + 1];

compute\_shift\_table(S, m, s);

int i := 0, j := m - 1;

**while** i ≤ n - m **do**

**while** t[i + j] = s[j] **do**

**if** j = 0 **then return** TRUE;

        j--;

    i := i + S[j];

    j := m - 1;

**return** FALSE;

---

# Wiederholte Vergleiche in übereinstimmenden Bereichen

- Nach dem Shift gibt es einen Bereich, in dem Übereinstimmung von  $s$  und  $t$  vorliegt.
- Allerdings werden auch in diesem Bereich wieder Vergleiche ausgeführt, da es letztendlich doch zu aufwendig ist, sich diesen Bereich explizit zu merken und bei folgenden Vergleichen von  $s$  in  $t$  zu überspringen.

# Ausgangssituation direkt nach Mismatch

- Der erste Mismatch soll im zu durchsuchenden Text  $t$  an der Stelle  $i + j$  (also im Suchwort  $s$  an Stelle  $j$ ) auftreten.
- Da der Boyer-Moore-Algorithmus das Suchwort von hinten nach vorne vergleicht, ergibt sich folgende Voraussetzung:

$$s_{j+1} \cdots s_{m-1} = t_{i+j+1} \cdots t_{i+m-1} \quad \wedge \quad s_j \neq t_{i+j}$$

- in Worten: Das Suffix des Suchworts ab Index  $j + 1$  stimmt mit den entsprechenden Zeichen im Text überein, das Zeichen an Position  $j$  jedoch nicht.

# Werte der Shift-Tabelle: kleine Shifts

- Um nun einen nicht nutzlosen Shift um  $\sigma$  Positionen zu erhalten, muss gelten:

$$s_{j+1-\sigma} \cdots s_{m-1-\sigma} = t_{i+j+1} \cdots t_{i+m-1} = s_{j+1} \cdots s_{m-1} \quad \wedge$$

$$s_j \neq s_{j-\sigma}$$

- Diese Bedingung ist nur für „kleine“ Shifts mit  $\sigma \leq j$  sinnvoll, also für solche, bei denen der Anfang des Suchworts  $s$  noch (mindestens) den gesamten bisher gematchten Bereich (inklusive das Mismatch-Zeichen) vom Text  $t$  abdeckt.
- Beispiel: erster Shift in der vorigen Abbildung

# Werte der Shift-Tabelle: große Shifts

- Für „große“ Shifts  $\sigma > j$  muss gelten, dass das Suffix des übereinstimmenden Bereichs mit dem Präfix des Suchwortes übereinstimmt, d.h.:

$$s_0 \cdots s_{m-1-\sigma} = t_{i+\sigma} \cdots t_{i+m-1} = s_\sigma \cdots s_{m-1}$$

- Zusammengefasst ergibt sich für beide Bedingungen:

$$\begin{aligned} \sigma \leq j & \wedge s_{j+1} \cdots s_{m-1} \in \mathcal{R}(s_{j+1-\sigma} \cdots s_{m-1}) & \wedge s_j \neq s_{j-\sigma} \\ \sigma > j & \wedge s_0 \cdots s_{m-1-\sigma} \in \mathcal{R}(s_0 \cdots s_{m-1}) \end{aligned}$$

$\mathcal{R}(s)$ : Menge aller Ränder von  $s$

# Zulässige und sichere Shifts

- Erfüllt ein Shift nun eine dieser Bedingungen, so nennt man diesen Shift **zulässig**.
- Um einen **sicheren Shift** zu erhalten, wählt man das minimale  $\sigma$ , das eine der Bedingungen erfüllt.
- Somit gilt:

$$S[j] = \min \left\{ \sigma : \begin{array}{l} (s_{j+1} \cdots s_{m-1} \in \mathcal{R}(s_{j+1-\sigma} \cdots s_{m-1}) \wedge \\ s_j \neq s_{j-\sigma} \wedge \sigma \leq j) \vee \\ (s_0 \cdots s_{m-1-\sigma} \in \mathcal{R}(s_0 \cdots s_{m-1}) \wedge \sigma > j) \vee \\ (\sigma = m) \end{array} \right\}$$

## Bestimmung der Shift-Tabelle

---

**Algorithmus 8** : `compute_shift_table(int S[], char s[], int m)`


---

```

for (int j := 0; j ≤ m; j++) do
  S[j] := m; // Initialisierung von S[]
// Teil 1:  $\sigma \leq j$ 
int border2[m + 1];
border2[0] := -1;
int i := border2[1] := 0;
for (int j' := 2; j' ≤ m; j'++) do
  // Hier gilt:  $i = \text{border2}[j' - 1]$ 
  while ((i ≥ 0) && (s[m - i - 1] ≠ s[m - j'])) do
    int  $\sigma := j' - i - 1$ ;
    S[m - i - 1] := min(S[m - i - 1],  $\sigma$ );
    i := border2[i];
  i++;
  border2[j'] := i;

```

## Bestimmung der Shift-Tabelle

---

**Algorithmus 9** : `compute_shift_table(int S[], char s[], int m)`

---

⋮

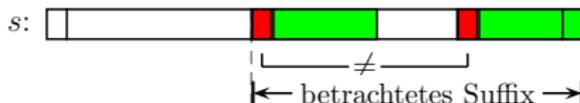
// Teil 2:  $\sigma > j$ `int j := 0;`**for** (`int i := border2[m]; i ≥ 0; i := border2[i]`) **do**    `int  $\sigma$  := m - i;`    **while** (`j <  $\sigma$` ) **do**        `S[j] := min(S[j],  $\sigma$ );`        `j++;`

# Bestimmung der Shift-Tabelle

- Zu Beginn wird die Shift-Tabelle an allen Stellen mit der Länge des Suchstrings initialisiert.
- Im Wesentlichen entsprechen beide Fälle von möglichen Shifts (siehe obige Vorüberlegungen) der Bestimmung von Rändern von Teilwörtern des gesuchten Wortes.
- Dennoch unterscheiden sich die hier betrachteten Fälle vom KMP-Algorithmus, da hier, zusätzlich zu den Rändern von Präfixen des Suchwortes, auch die Ränder von Suffixen des Suchwortes gesucht sind.

Bestimmung der Shift-Tabelle:  $\sigma \leq j$ 

- Dies gilt besonders für den ersten Fall ( $\sigma \leq j$ ). Hier soll das Zeichen unmittelbar vor dem Rand des Suffixes ungleich dem Zeichen unmittelbar vor dem gesamten Suffix sein:



- Diese Situation entspricht dem Fall in der Berechnung eines eigentlichen Randes, bei dem der vorhandene Rand nicht zu einem längeren Rand fortgesetzt werden kann (nur werden hier Suffixe statt Präfixe betrachtet).
- Daher sieht die erste for-Schleife genau so aus, wie in der Prozedur `compute_borders` des KMP-Algorithmus.
- Lässt sich ein betrachteter Rand nicht verlängern, so wird die while-Schleife ausgeführt.

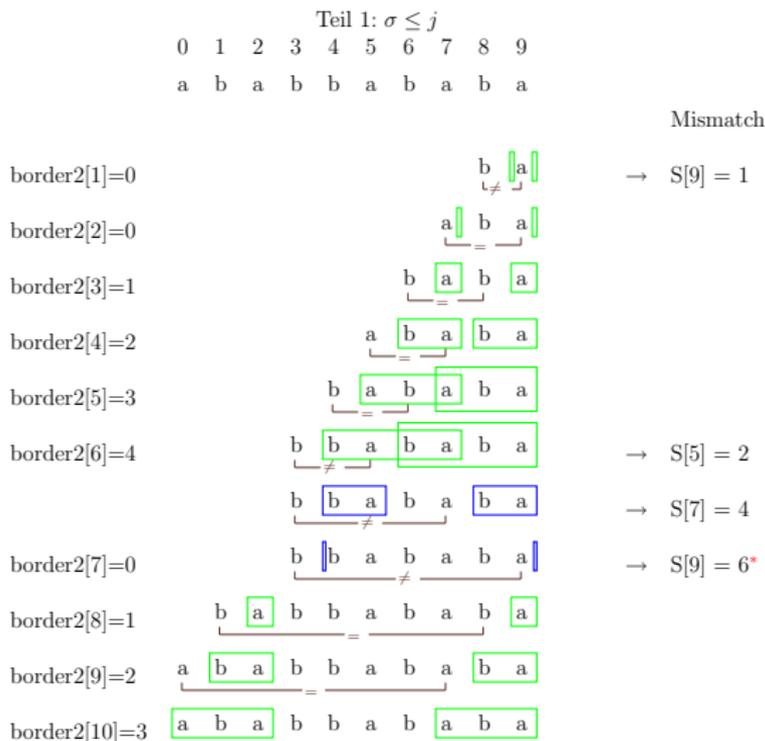
## Bestimmung der Shift-Tabelle: $\sigma > j$

- Im zweiten Fall ( $\sigma > j$ ) müssen wir alle Ränder von  $s$  durchlaufen.
- Der längste Rand ungleich  $s$  ist der eigentliche Rand von  $s$ . Diesen erhalten wir über  $border2[m]$ , da ja der Suffix der Länge  $m$  von  $s$  gerade wieder  $s$  ist.
- Der nächstkürzere Rand von  $s$  muss ein Rand des eigentlichen Randes von  $s$  sein. Damit es der nächstkürzere Rand ist, muss  $s$  der eigentliche Rand des eigentlichen Randes von  $s$  sein, also das Suffix der Länge  $border2[border2[s]]$ .
- Somit können wir alle Ränder von  $s$  durchlaufen, indem wir immer vom aktuell betrachteten Rand der Länge  $\ell$  das Suffix der Länge  $border2[\ell]$  wählen. Dies geschieht in der for-Schleife im zweiten Teil. Solange der Shift  $\sigma$  größer als die Position  $j$  eines Mismatches ist, wird die Shift-Tabelle aktualisiert. Dies geschieht in der inneren while-Schleife.

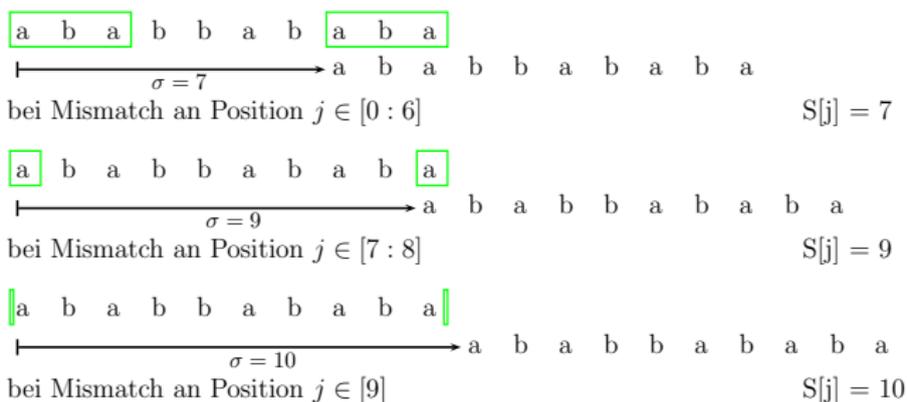
# Bestimmung der Shift-Tabelle

- Bei der Aktualisierung der Shift-Tabelle werden nur zulässige Werte berücksichtigt. Damit sind die Einträge der Shift-Tabelle (d.h. die Länge der Shifts) nie zu klein.
- Eigentlich müsste jetzt noch gezeigt werden, dass die Werte auch nicht zu groß sind, d.h. dass es keine Situation geben kann, in der ein kleinerer Shift möglich wäre.
- Dass dies nicht der Fall ist, kann mit einem Widerspruchsbeweis gezeigt werden (man nehme dazu an, bei einem Mismatch an einer Position  $j$  in  $s$  gäbe es einen kürzeren Shift gemäß der Strong-Good-Suffix-Rule und leite daraus einen Widerspruch her).

## Bestimmung der Shift-Tabelle: Beispiel



## Bestimmung der Shift-Tabelle: Beispiel

Teil 2:  $\sigma > j$ 

Zusammenfassung:

$S[0] =$	7	7	$S[5] =$	2	2
$S[1] =$	7	7	$S[6] =$	7	7
$S[2] =$	7	7	$S[7] =$	4	4
$S[3] =$	7	7	$S[8] =$	9	9
$S[4] =$	7	7	$S[9] =$	1	1

1. Teil   2. Teil   Erg

1. Teil   2. Teil   Erg